Practical Algorithms for Performance Guarantees in Buffered Crossbars

Shang-Tse Chuang, Sundar Iyer, Nick McKeown
Computer Systems Laboratory,
Stanford University,
Stanford, CA 94305-9030.
{stchuang, sundaes, nickm}@stanford.edu

***Abstract*— This paper is about high capacity switches and routers that give guaranteed throughput, rate and delay guarantees. Many routers are built using input queueing or combined input and output queueing (CIOQ), using crossbar switching fabrics. But such routers require impractically complex scheduling algorithms to provide the desired guarantees. We explore how a buffered crossbar — a crossbar switch with a packet buffer at each crosspoint — can provide guaranteed performance (throughput, rate, and delay), with less complex, practical scheduling algorithms. We describe scheduling algorithms that operate in parallel on each input and output port, and hence are scalable. With these algorithms, buffered crossbars with a speedup of two can provide 100% throughput, rate, and delay guarantees.**

***Index Terms*— system design, packet switching, buffered crossbar, scheduling algorithm, performance guarantees, throughput, quality of service.**

I. Background

Network operators want high capacity routers that give guaranteed performance. First, they prefer routers that guarantee throughput so they can maximize the utilization of their expensive long-haul links. Second, they want routers that can allocate to each flow a guaranteed rate. Third, they would like the capability to control the delay for packets of individual flows for real-time applications. Because they want high capacity, the trend has been towards input queued or combined input and output queued (CIOQ) routers. Most of these routers use a crossbar switching fabric with a centralized scheduler. While it is theoretically possible to build crossbar schedulers that give 100% throughput [1] or rate and delay guarantees [2][3] they are considered too complex to be practical. No commercial backbone router today can make hard guarantees on throughput, rate or delay.

In practice, commercial systems use heuristics such as *i*SLIP [5] or a maximal matching algorithm such as WFA [6] with insufficient speedup to give guarantees. Perhaps the most promising way of obtaining guaranteed performance has been to use maximal matching with a speedup of two in the switch fabric [7]. But this only gives guaranteed throughput with no guarantees on rates and delay. And it still requires a centralized scheduler which doesn't scale with an increase in the number of ports due to the communication complexity. In this paper we'll see that the scheduler for a buffered crossbar is much simpler, and therefore more scalable, than for a traditional unbuffered crossbar.

A. The Buffered Crossbar

Figure 1 shows a 3×3 buffered crossbar, with line-rate R. To prevent head-of-line blocking, the inputs maintain virtual output queues ($VOQs$). Fixed length packets[1] wait in the $VOQs$ to be transferred across the switch. Each crosspoint contains a buffer that can hold one cell. The buffer between input i and output j is denoted as B_{ij}; when the buffer holds a cell, $B_{ij} = 1$, else $B_{ij} = 0$.

Because the packets are all the same length, time is slotted, with a time slot equal to the time it takes for a cell to arrive on the external line. Internally, the switch runs faster than the external line, and the ratio between the two is the *speedup*. If the switch can remove S cells from each input and transfer S cells to each output in a time slot, then it has a speedup of S. Throughout most of this paper we will assume that $S = 2$, and so the switch has output queues.

B. Why use Buffered Crossbars?

Buffered crossbars are interesting because they have simpler scheduling algorithms than an unbuffered crossbar. In an unbuffered crossbar, the scheduler must find a matching between inputs and outputs that doesn't oversubscribe either. Overcoming both constraints at the same time leads to complex scheduling algorithms, such as maximal [6], maximum size [8] and maximum weight bipartite matchings [1], or iterative schedulers that are hard to pipeline [5, 9].

[1]We will assume that variable length packets are processed internally as fixed length cells. This is common practice in high performance LAN switches and routers; variable length packets are segmented into cells as they arrive, carried across the switch as cells, and reassembled back into packets again before they depart. The size of the cell doesn't matter here, and is not necessarily equal in length to a 53-byte ATM cell.

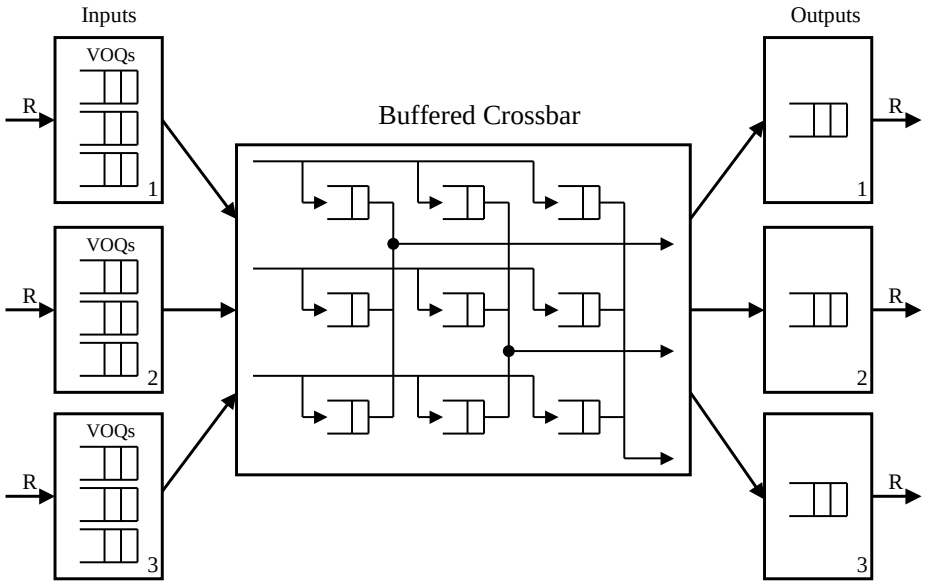

Fig. 1. The architecture of the buffered crossbar with three ports.

The appeal of a buffered crossbar switch is that its scheduler is much simpler. The scheduler operates in two stages. First, each input (independently and in parallel) picks a cell to place into a crosspoint. And then in the second stage each output (independently and in parallel) picks a crosspoint to take a cell from. The processing can be distributed to run on each input and output, and so no longer requires a single centralized scheduler. It can be pipelined to run at high speed, making buffered crossbars appealing for high performance switches and routers.

Researchers first noticed via simulation that buffered crossbars provide good throughput for admissible uniform traffic with simple algorithms [24, 10, 11, 12]. Simulations also indicated that, with modest speedup, a buffered crossbar can closely approximate fair queueing [13]. In [23], the authors described a mechanism to provide fair allocation and confirmed through simulations that a buffered crossbar can allocate service in a weighted max-min fair manner. Until recently, there were no analytical results on guaranteed throughput to explain or confirm the observations made by simulations.

The first analytical results came in 2001, when Javidi *et al.* proved that, with uniform traffic, a buffered crossbar can achieve 100% throughput [14]. More recently, Magill *et al.* proved that a buffered crossbar with a speedup of two can mimic[2] a first come first serve output queued (FCFS-OQ) switch with any arrival traffic pattern [15]. Magill *et al.* also showed that a buffered crossbar with k cells per crosspoint can mimic a FCFS-OQ switch with k strict priorities.

In this paper, we describe a series of algorithms with a broad class of performance guarantees over and above FCFS and strict priority FCFS emulation. We prove that these algorithms can achieve 100% throughput, can mimic an OQ switch using a weighted round robin scheduler (which gives rate guarantees), and can also achieve delay guarantees. The main benefit of these algorithms is that each input and output makes simple scheduling decisions *independently* and in *parallel*, eliminating the need for a centralized scheduler. Our results show buffered crossbars can greatly simplify the scheduling process.

Of course, simplifying the scheduler comes at the expense of a more complicated crossbar; it now has to hold and maintain N^2 packet buffers. In the past this would have been prohibitively complex: The number of ports and capacity of a crossbar switch used to be limited by the N^2 crosspoints that dominated the chip area (hence the development of multi-stage switch fabrics, such as Clos, Banyan and Omega switches based on smaller crossbar elements). But nowadays, crossbar switches are limited by the number of pins required to get data on and off the chip [17]. Improvements in process technology, and reductions in geometries, means that the logic required for N^2 crosspoints is small compared to the size of chip needed for N inputs and N outputs. The chips are pad-limited, with an underutilized die. A buffered crossbar can use the unused die for packet buffers. We believe that in current technology, the 128×128 unbuffered crossbar switch reported in [17] could hold 128^2 cell buffers.

C. Organization of this paper

The rest of the paper is organized as follows. In Section II, we show that a buffered crossbar can give 100% throughput when $S = 2$. In Section III, we briefly

[2]Two different switches are said to mimic [2, 16] each other, if under identical inputs, identical packets depart from each switch at the same time.

review the counting method introduced in [2] used to show how a CIOQ switch can mimic an OQ switch. In Section IV we show how the counting method can be applied to a buffered crossbar and so as to mimic a class of OQ switches. In Section V, we describe how a buffered crossbar can give rate guarantees between an input/output pair for a weighted round robin scheduler. In Section VI, we show that the buffered crossbar with one cell per crosspoint can give delay guarantees when $S = 3$, and introduce a novel mechanism called header scheduling which supports delay guarantees when $S = 2$.

II. Achieving 100% throughput with an arbitrary scheduling algorithm

Figure 2 shows the scheduling phases in a buffered crossbar with a speedup of two. The two scheduling phases each consists of two parts: input scheduling, and output scheduling. In the input scheduling phase, each input (independently and in parallel) picks a cell to place into an empty crosspoint. In the output scheduling phase, each output (independently and in parallel) picks a cell from a non-empty crosspoint to take from. The key to creating a scheduling algorithm is determining the input and output scheduling policy which decides how inputs and outputs pick cells in the scheduling phases. We will see a number of different policies each of which provides a different scheduling algorithm. The first algorithm we'll consider is the most general. In each scheduling phase, the input picks any non-empty VOQ, and the output picks any non-empty crosspoint.

We will adopt the following notation and definitions. The switch has N ports, and VOQ_{ij} holds cells at input i destined for output j. Z_{ij} is the occupancy of VOQ_{ij},[3] and $\tilde{Z_{ij}} = Z_{ij} + B_{ij}$ is the sum of the number of cells in the VOQ and the corresponding crosspoint. We will assume that all arrivals to input $i \in 1, 2, 3, ..., N$ are Bernoulli i.i.d. with rate λ_i, and are destined to each output $j \in 1, 2, 3, ...N$ with probability λ_{ij}. We will denote the arrival matrix as, $A \equiv [\lambda_{ij}]$, where

$$\lambda_i = \sum_{j=1}^{N} \lambda_{ij}, \lambda_j = \sum_{i=1}^{N} \lambda_{ij}, \lambda_{ij} \geq 0$$

Definition 1: An arrival process is said to be admissible when no input or output is oversubscribed, i.e., when

$$\sum_{i=1}^{N} \lambda_{ij} < 1, \sum_{j=1}^{N} \lambda_{ij} < 1$$

Definition 2: **100% Throughput —** An algorithm is said to give 100% throughput iff for any admissible Bernoulli iid traffic, the queue sizes are finite, i.e., $\forall i, j$,

$$\lim_{n\to\infty} \sup Z_{ij}(n) < \infty, w.p.1$$

[3]We'll see later that other queueing structures are useful and it is not necessary to place cells in $VOQs$.

where Z_{ij} corresponds to the occupancy of VOQ_{ij}.

In what follows, we will show that the buffered crossbar can give 100% throughput. The result is quite strong in the sense that it holds for any arbitrary work-conserving input and output scheduling policy with a speedup of two. In other words, each input i can choose to serve any non-empty VOQ for which $B_{ij} = 0$, and each output j can choose to serve any crosspoint for which $B_{ij} = 1$.

Theorem 1: (Sufficiency) A buffered crossbar can achieve 100% throughput with speedup two for any Bernoulli i.i.d. admissible traffic.

Proof: We describe an intuition of the proof. The main proof appears in Appendix A.

For each VOQ_{ij}, let C_{ij} denote the sum of the cells waiting at input i and the cells waiting at all inputs destined to output j (including cells in the crosspoints for output j),

$$C_{ij} = \sum_{k} Z_{ik} + \sum_{k} (Z_{kj} + B_{kj}) \quad (1)$$

It is easy to see that when VOQ_{ij} is non-empty (i.e. $Z_{ij} > 0$), then C_{ij} decreases in every scheduling phase. There are two cases:

- *Case 1:* $B_{ij} = 1$. Output j will receive one cell from the buffers destined to it and $\sum_k (Z_{kj} + B_{kj})$ will decrease by one.
- *Case 2:* $B_{ij} = 0$. Input i will send one cell from its $VOQs$ to a crosspoint, and $\sum_k Z_{ik}$ will decrease by one.[4]

With $S = 2$, C_{ij} will decrease by two per time slot. When the inputs and outputs are not oversubscribed, the expected increase in C_{ij} is strictly less than two per time slot. So the expected change in C_{ij} is negative over the time slot, and this means that the expected value of C_{ij} is bounded. This in turn implies that the expected value of Z_{ij} is bounded and the buffered crossbar has 100% throughput. ■

III. Background of the Counting Method

An OQ switch can provide throughput, rate and delay guarantees. In [2], the authors used a counting method to show that a CIOQ switch using a traditional unbuffered crossbar with a speedup of two can mimic an OQ switch and therefore also provide rate and delay guarantees. The method in [2] was to show that a CIOQ switch with speedup of two can mimic a so-called Push-In First-Out (PIFO) queueing policy. PIFO is a general queueing policy that includes WFQ (weighted fair queueing) as a

[4]If a cell from VOQ_{ij} is sent to crosspoint B_{ij}, then $\sum_k (Z_{kj} + B_{kj})$ stays the same at the end of the input scheduling phase since Z_{ij} decreases by one and B_{ij} increases by one. In the output schedule, *Case 1* applies and C_{ij} will further decrease by one. As a result, if a cell from VOQ_{ij} is sent to crosspoint B_{ij}, then C_{ij} decreases by two in that scheduling phase.

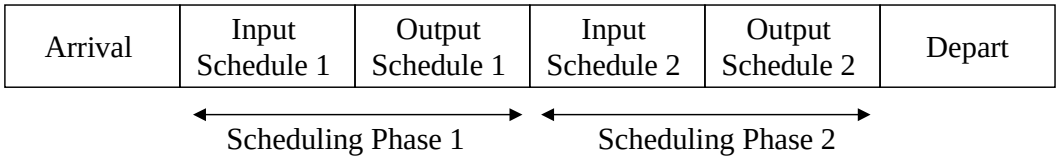

Fig. 2. The scheduling phases for the buffered crossbar. The exact order of the phases doesn't matter; but we will use this order to simplify proofs.

special case. If a CIOQ switch can implement a PIFO policy, then it can implement WFQ which means it can support rate and delay guarantees [18, 19].

In a PIFO queue, there is a single queue of all cells waiting to depart from an output. When a new cell arrives for the output, it is "pushed-in" to an arbitrary location in the queue. Once in the queue, the cell's relative ordering with cells already in the queue doesn't change; cells can't switch places. Of course, new cells can arrive later and get pushed-in ahead of (or behind) the cell. Cells can only depart from head of line.

In order to mimic a PIFO-OQ switch, a cell must be transferred to the output of the CIOQ switch before the departure time of the counterpart cell in the OQ switch. If the cell is prevented from reaching its output in time, the departures from both switches is not identical, and we'll fail to mimic the OQ switch. Selecting the appropriate scheduling algorithm is the key to ensuring that the cell will reach the output of the CIOQ switch in time.

The general structure of the scheduling algorithm described in [2] assumed that each input of the CIOQ switch maintains a priority list, which can be thought of as an ordered set of cells waiting at the input port. A cell can be prevented from reaching its output on time by other cells at its input with a higher priority. The more cells ahead of it in the priority list, the longer it might take to be transferred to the output. Many orderings of the cells are possible — each ordering leading to a different switch scheduling algorithm.

In addition, each output also maintains an output priority list: an ordered list of cells at the inputs waiting to be transferred to a particular output. The output priority list is constructed based on the order in which the cells would depart from the OQ switch. This priority list will depend on the queueing policy followed by the OQ switch (i.e., WFQ, strict priorities, etc.).

The following definitions previously defined in [2] is necessary for the understanding of the rest of the paper.

Definition 3: **Output Cushion —** At any time, the output cushion of a cell c, $OC(c)$, is the number of cells at c's output port that has an earlier departure order than cell c.

Definition 4: **Input Thread —** At any time, the input thread of cell c, $IT(c)$, is the number of cells at c's input with a higher priority.

Definition 5: **Slackness —** At any time, the slackness of cell c, $L(c)$, equals its output cushion minus its input thread; i.e., $L(c) = OC(c) - IT(c)$.

The slackness reflects the urgency with which we must transfer the cell to its output. The key to identical behavior is to find scheduling algorithms for which the slackness is always non-negative. Although not strictly necessary, this will ensure that when a cell is transferred to the output its output cushion is non-negative, (or reaches zero in the time slot it is transferred). The idea is that when the output cushion of a cell reaches zero, the input thread of that cell must also equal zero. This means that either: (1) the cell is already at its output, and will depart the output on time, or (2) the cell is at the head of its priority list (because its input thread is zero), and will be transferred to the output immediately, which ensures that the cell will depart the output on time.

Based on the input and output priority lists, the counting method required that in each scheduling phase, at least one of the following conditions for each cell c is satisfied: (1) cell c is transferred from the input side, (2) a cell that is ahead of cell c in its input priority list is transferred from the input side, or (3) a cell that is ahead of cell c in its output priority list is transferred to the output side.

In [2], it was proved that meeting the conditions of the counting method ensured that the slackness of a cell increased by at least one in each scheduling phase, which was essential in proving that the slackness of any cell is always non-negative. However, [2] required a *stable marriage algorithm* [4] to meet the conditions of the counting method. The solution required up to at least N iterations of the stable marriage algorithm for a switch with N ports, making them too complex to implement for fast switches with large number of ports.

We will now show how the buffered crossbar can also meet the conditions of the counting method in a simple distributed manner where each input and output makes decisions independently and in parallel.

IV. Counting Method with a Buffered Crossbar

In order to ensure that the slackness of a cell increases by at least one in each scheduling phase for a buffered crossbar, the input and output scheduling policies must carefully be selected to guarantee that the conditions of the counting method are met. The input

scheduling policy gives preference to cells based on the input priority list. Similarly, the output scheduling policy gives preference to cells based on the output priority list. Since the output priority list is ordered based on departure order, preference is given to cells with an earlier departure order.

However, the buffered crossbar has an additional requirement to meet the conditions of the counting method. The input priority list also must be arranged so cells destined to the same output are ordered based on departure order. Specifically, cells to the same output with an earlier departure order must have a higher priority. Cells to different outputs can still be ordered in any way. This requirement is necessary to ensure that, in the output scheduling phase, the cell selected has the earliest departure order of the cells stored in the input queues corresponding to the non-empty crosspoints, as can be seen in the following example.

Let cells a, b, and c all be destined to output j. Cell a is stored in input queue i_1, cell b is stored in input queue i_2, cell c is stored in crosspoint $B_{i_1 j}$, and no other cells are destined to output j at time t. The departure order is $t_a < t_b < t_c$ for cells a, b, and c respectively. In the input scheduling phase, input i_1 does not select cell a since cell c is already in the crosspoint $B_{i_1 j}$ and input i_2 selects cell b. In the output scheduling phase, cell b is selected since it has an earlier departure order than cell c. As a result, the conditions of the counting method is not met for cell a since cell b which has a later departure order does not have a higher priority than cell a in the output priority list. This occurred because at some point in time cell c was incorrectly given a higher priority than cell a in the input priority list. This motivates the following "Group By Virtual Output Queue" insertion policy previously described in [2].

GBVOQ Insertion Policy:

1) When a cell arrives to a non-empty VOQ, the cell is inserted in the input priority list just behind the last cell belonging to the same VOQ. This ensures that cells destined to the same output are ordered based on departure order.
2) When a cell arrives to an empty VOQ, the cell is inserted at the head of the input priority list.

At first glance, it seems unfair to insert a cell which arrives to an empty VOQ at the head of the input priority list. However, it is possible that there are no other cells in the system destined to that output. Therefore, the cell may immediately need to be transferred to the output in order to keep that output busy.

We will now prove in the following lemma that the buffered crossbar can satisfy the conditions of the counting method.

Lemma 1: The slackness $L(c)$ of a cell c decreases by at least one in each scheduling phase.

Proof: Let's assume that cell c belongs to VOQ_{ij}. There are two cases to consider in a scheduling phase.

- *Case 1:* If $B_{ij} = 0$, then we know that in the input scheduling phase, a cell will be transferred from input i to one of the buffers B_{i*}. If cell c is transferred to B_{ij}, then we no longer need to consider it.[5] If a different cell is transferred to its crosspoint, the cell would belong to c's input thread, and $IT(c)$ will decrease by one.
- *Case 2:* If $B_{ij} = 1$, then a cell will be transferred from one of the crosspoints B_{*j} to output j in the output scheduling phase. By definition of the GBVOQ insertion policy the cell in the crosspoint B_{ij} has an earlier departure order than cell c. Since the output scheduling policy selects the non-empty crosspoint that contains the cell with the earliest departure order, $OC(c)$ increases by one.

Therefore, $L(c)$ increases by at least one per scheduling phase. ■

The counting method using the GBVOQ insertion policy can be applied trivially to show that a buffered crossbar can mimic a restricted PIFO-OQ switch, i.e., a PIFO-OQ switch with the restriction that cells from an input/output pair depart the switch in the order they arrive. This restricted policy includes output link schedulers which are fair across all inputs, i.e., provide rate guarantees between each input/output pair.[6]

Theorem 2: (Sufficiency) A buffered crossbar with a speedup of two can mimic the restricted PIFO-OQ switch, regardless of the incoming traffic pattern.

Proof: See Appendix B. ■

V. Rate Guarantees with a Buffered Crosbar

Our goal is to find a way for a buffered crossbar to provide a pre-determined and guaranteed rate for each flow passing through the switch. In an OQ switch, this is straightforward to do with, for example, a weighted round-robin (WRR) scheduler. A WRR scheduler serves each flow queue in turn in round-robin order, giving service to each queue in proportion to the weight assigned to it. If a queue is empty, it is skipped and not served. It is well known that — when packets are all of equal length — WRR gives each flow a rate in proportion to its weight, and hence can give a minimum rate guarantee to each flow. Furthermore, if the arrival processes are suitably constrained (e.g. by leaky buckets), then the delay of each packet through the switch can be bounded [19].

One approach to proving that a buffered crossbar can provide rate guarantees would be to show that the

[5] If a cell is transferred to the crosspoint then it is available for selection in the output scheduling phase and can be placed on the output line whenever necessary.

[6] FCFS is an example of this restricted policy and is extensively analyzed in [15].

buffered crossbar can emulate a PIFO-OQ switch. Then, because WRR is a special case of PIFO, we can conclude that the buffered crossbar can support WRR and provide rate guarantees just as an OQ switch can. But as we'll see in the next section, it is harder for a buffered crossbar to emulate a PIFO-OQ switch.

So instead, we're going to start by solving an easier problem by considering a restricted PIFO-OQ switch. In what follows, a flow is defined to be those cells between a specific input/output pair. We'll consider a WRR-OQ switch that serves these flows in WRR order; and try to emulate the same behavior with a buffered crossbar.

In the WRR-OQ switch, each arriving cell is assigned a virtual finishing time by the WRR scheduler. Reviewing the way a WRR scheduler works, consider the cells at just one output. The k^{th} cell from input i is assigned virtual finishing time F_i^k, where

$$F_i^k = max\{F_i^{k-1}, V(a_i^k)\} + \frac{1}{\phi_i}.$$

a_i^k is the arrival time of the k^{th} cell from input i to that output, $V(t)$ is the virtual time (round number) at time t, and ϕ_i is the weight assigned to input i. When the output line is free, the WRR scheduler serves the cell with the smallest virtual finish time.

Similarly, in the WRR buffered crossbar, a virtual finishing time needs to be assigned to each cell, so as to determine the correct departure order. The problem is that cells are buffered at both inputs and outputs (and in the crossbar). Calculating the virtual finishing time when the cell arrives would require the input to have information about the cell's output, and all the cells at other inputs destined to it. This is impractical. Fortunately with our restricted definition of flows, cells are held in the input priority list in their departure order and — as we'll show below — it is sufficient for the output to assign a virtual finish time only when cells reach the crosspoint.

The output needs to know upon arrival of the k^{th} cell to the switch whether the $k-1^{th}$ cell (from this input to the given output) has departed. If it has departed, then the k^{th} cell is transferred to the crosspoint immediately and is assigned the virtual finish time based only on the current virtual time. If it has not departed, then the k^{th} cell is not be transferred immediately or the $k-1^{th}$ cell must be in the output queue. Therefore, the k^{th} cell is assigned the virtual finish time based on the virtual finish time of the $k-1^{th}$ cell. We will formalize these cases in the proof.

Theorem 3: (Sufficiency) A buffered crossbar can mimic an OQ switch using a weighted round-robin policy with speedup two, regardless of the incoming traffic pattern.

Proof: See appendix C. ■

A consequence of the proof is that the scheduling of cells to emulate WRR service is practical, and can be done independently and in parallel by each input and output. An input independently generates its input priority list with only local knowledge. It picks a cell to place in the crosspoint knowing only which crosspoints are empty. An output calculates the virtual finish time of a cell when the cell arrives to the crosspoint. The output just needs to know if the cell was transferred to the crosspoint immediately upon arrival, which can be carried in one bit of the cell. Each output picks the cell in the crosspoint with the smallest virtual finish time.

VI. Delay Guarantees in a Buffered Crossbar

In practice, an input/output pair carries many flows, not just one. For example, it carries TCP flows between source/destination pairs, and we might want to give each flow a different rate or delay guarantee. In order to do this, we need to relax our constraint on the definition of the flow, and determine how to assign a different rate to each flow. This is what we will do next; and we'll see that it increases the complexity of the buffered crossbar and requires more speedup.

In a PIFO-OQ switch, an arriving cell can be pushed into any location in the queue. It could, for example, be scheduled to depart ahead of all currently queued cells between the same input/output pair. In order to meet the conditions of the counting method, the cell in the crosspoint must have the earliest departure order of all cells stored in the input queue belonging to its input/output pair. This causes problems for the buffered crossbar switch. Imagine the situation in which a crosspoint has a cell in it, and an arriving cell has an *earlier* departure order than the cell in the crosspoint. This causes what we call *"crosspoint blocking"* since the arriving cell cannot overtake the cell in the crosspoint.

If each crosspoint had a cell buffer for each flow, crosspoint blocking could be avoided.[7] However, this doesn't scale for a large number of flows. We now show how a buffered crossbar can overcome crosspoint blocking in a manner which is independent of the number of flows between an input/output pair.

A. Delay guarantees with speedup three

When a cell arrives to an input with an earlier departure order than the cell in the crosspoint buffer, we will swap the cell in the crosspoint with the newly arriving cell. Logically, the cell that was previously in the crosspoint is recalled to the input where it is treated like a newly arriving cell. By modifying the arrival phase to include swapping, crosspoint blocking can be avoided.

[7] [15] showed a buffered crossbar with k cells per crosspoint could mimic an OQ switch with k strict priority flows between each input/output pair.

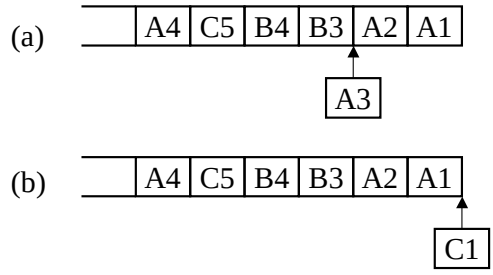

Fig. 3. The insertion policy for achieving delay guarantees. The figure shows the priority list for a given input. The letter denotes the output destination, and the number denotes the cell's departure order for a given output. (a) Arriving cell $A3$ is inserted immediately after cell $A2$. (b) Arriving cell $C1$ is inserted at the head of the priority list since no other cell has a departure order less than cell $C1$ destined to output C.

This is at the expense of additional speedup to perform the swap.

The modified arrival phase requires a new insertion policy. This policy needs to meet two requirements: (1) To prevent crosspoint blocking, cells from an input/output pair must be inserted based on their departure order. (2) The slackness of a cell must be non-negative when inserted.

The Insertion Policy: As a consequence of the first requirement, an arriving cell c destined to output port j is inserted behind all cells destined to output j with a departure order less than cell c. To satisfy the second requirement, cell c is inserted *immediately* behind the cell that departs before it (if it exists), destined to the same output. If no such cell exists, cell c is inserted at the head of the priority list. This ensures that the slackness of the cell c is non-negative.

The priority list defined by this insertion policy has the property that cells from input i to output j are ordered based on their PIFO departure order. An example is shown in Figure 3.[8]

Theorem 4: (Sufficiency) A buffered crossbar can mimic a PIFO-OQ switch (and hence give delay guarantees) with speedup three, regardless of the incoming traffic pattern.

Proof: See Appendix D. ■

B. Delay guarantees with speedup two

We overcame crosspoint blocking by swapping the cell in a crosspoint with a newly arriving cell. This was necessary because we allowed cells to be transferred to the buffered crossbar even before they were scheduled to depart. This early transfer was the cause of crosspoint blocking, and thus required swapping. But we could eliminate the need for swapping if we avoided transferring a cell to the crosspoint until it was really ready to be transferred to the output.

For example, we could put the cell header in the crosspoint buffer, and only transfer the cell across the crossbar when chosen by the output. Scheduling would now be done in two distinct phases. First, input and output scheduling would be done based on the cell headers. Second, the cell bodies would be transferred when the output chooses a cell. We call this *header scheduling*.

However, this creates a problem. An input could receive up to N grants (one from each output) in a single output scheduling phase. Fortunately, over p consecutive phases the number of grants received by an input is bounded by $p + N - 1$. This is because an input can communicate at most one header per input scheduling phase, and there are at most N outstanding headers (one for each crosspoint) per input. On the other hand, each output grants at most one header per scheduling phase. So there are at most p grants for an output over any p consecutive scheduling phases.

Since inputs send at most one cell per scheduling phase and an input can receive up to $p + N - 1$ grants in p consecutive scheduling phases, the cell for a header granted in phase x might not be transferred until phase $x + N - 1$. With only one cell per crosspoint, a cell can prevent an input from sending another cell to the same output for another $N - 1$ scheduling phases. We therefore modify the buffered crossbar to have N cells of buffering per output as shown in Figure 4. There are still N^2 buffers in the crossbar, but buffers are dedicated to outputs, rather than input/output pairs.

Theorem 5: (Sufficiency) With speedup two and N cells of buffering per output, a buffered crossbar can mimic a PIFO-OQ switch with a fixed delay of $N/2$ time slots.

Proof: See appendix E. ■

The result comes at the expense of a more complicated buffering scheme in the crossbar and requires N cells buffering per output. Since these N cells can arrive in the same scheduling phase, there is an additional implementation complexity. This can be eliminated by modifying the buffered crossbar so it has N cells for each B_{ij}, for a total of N^3 cells. In the latter case, no more than one cell can arrive to a crosspoint in each scheduling phase. While requiring more storage, it will also mimic a PIFO-OQ switch with a fixed delay of $N/2$ time slots with speedup two. This might be practical for small values of N. In both modified buffered crossbar architectures, the number of crosspoints is independent of the number of flows in the switch.

[8]This insertion policy should be contrasted with the CCF insertion policy in [2], which does not maintain cells from input i to output j in the correct departure order. CCF does not need to maintain this ordering because the stable marriage problem described in [2] considers all cells queued at the inputs when scheduling.

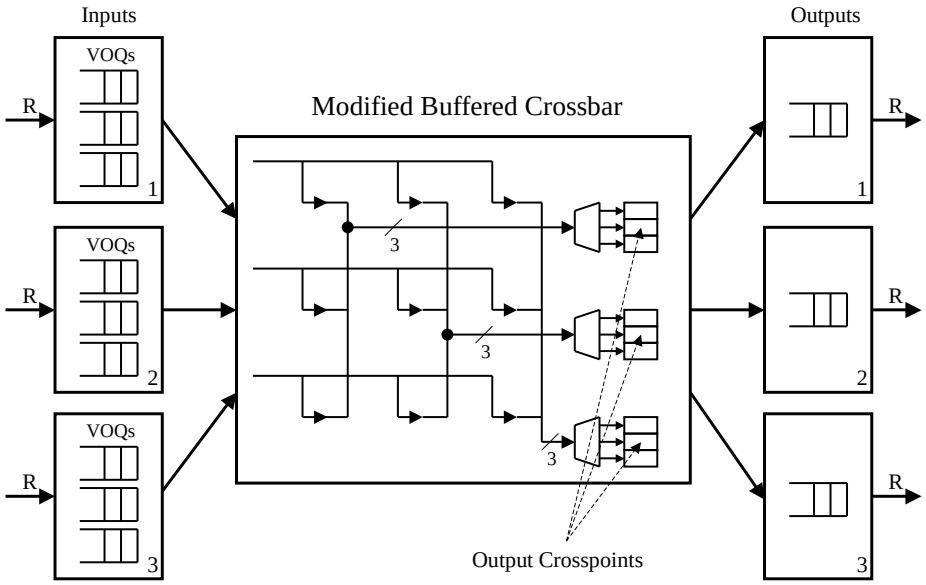

Fig. 4. The architecture of a modified buffered crossbar with three ports.

VII. Conclusions

It is hard to scale crossbar-based routers because the scheduler for a crossbar must resolve the input and output constraints simultaneously. Whereas centralized schedulers get very complicated, the scheduler for a buffered crossbar allows inputs and outputs to make decisions independently and in parallel. With speedup two, and scheduling algorithms which are distributed and easy to implement, buffered crossbars provide throughput, rate and delay guarantees.

Although the crossbar is more complex than before, the bandwidth and pin count is the same as before, the CIOQ architecture is maintained, and no memory needs to run faster than twice the line-rate. This provides a simple path to scale crossbar based routers.

References

[1] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% Throughput in an Input-Queued Switch," *IEEE Transactions on Communications*, Vol. 47, No. 8, Aug. 1999.

[2] S-T. Chuang, A. Goel, N. McKeown, and B. Prabhakar, "Matching Output Queueing with a Combined Input Output Queued Switch", *IEEE J. Select. Areas Commun.*, 17, No. 6, pp. 1030–1039. A short version appears in *The Proceedings of Infocom '99*.

[3] S. Iyer, R. Zhang, and N. McKeown, "Routers with a Single Stage of Buffering", *ACM SIGCOMM '02*, Pittsburgh, USA, Sep. 2002.

[4] D. Gale, and L.S. Shapley, "College Admissions and the Stability of Marriage," *American Mathematical Monthly*, Vol. 69, pp. 9–15, 1962.

[5] N. McKeown, "iSLIP: A Scheduling Algorithm for Input-Queued Switches" *IEEE Transactions on Networking*, Vol. 7, No. 2, Apr. 1999.

[6] Y. Tamir, and H.C. Chi, "Symmetric crossbar arbiters for VLSI communication switches," *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 1, pp. 13–27, 1993.

[7] J. Dai, and B. Prabhakar, "The throughput of data switches with and without speedup", *IEEE INFOCOM 2000*, Vol. 2, p. 556–564, Tel Aviv, Mar. 2000.

[8] J. E. Hopcroft, R. M. Karp, "An $O(N^{5/2})$ Algorithm for Maximum Matching in Bipartite Graphs," *Society for Industrial and Applied Mathematics J. Computers*, Vol. 2, pp. 225–231.

[9] T.E. Anderson, S.S. Owicki, J.B. Saxe, and C.P. Thacker, "High speed switch scheduling for local area networks," *ACM Transactions on Computer Systems*, Vol. 11, No. 4, pp. 319–352, Nov. 1993.

[10] Rojas-Cessa, E. Oki, Z. Jing, and H. J. Chao, "CIXB-1: Combined Input-One-cell-Crosspoint Buffered Switch," *IEEE Workshop on High Performance Switching and Routing*, Dallas, TX, July 2001.

[11] Rojas-Cessa, E. Oki, and H. J. Chao, "CIXOB-k: Combined Input-Crosspoint-Output Buffered Packet Switch," in *IEEE Globecom*, San Antonio, Texas, Nov. 2001.

[12] Lotfi Mhamdi, Mounir Hamdi, "MCBF: A High-Performance Scheduling Algorithm for Buffered Crossbar Switches", IEEE Communications Letters, 2003.

[13] D. Stephens, H. Zhang, "Implementing Distributed Packet Fair Queueing in a Scalable Switch Architecture", in *Proc. INFOCOM* 1998.

[14] T. Javidi, R Magill, and T. Hrabik, "A High Throughput Scheduling Algorithm for a Buffered Crossbar Switch Fabric", in *Proc. IEEE International Conference on Communications*, Vol. 5, pp. 1586–1591

[15] B. Magill, C. Rohrs, R. Stevenson, "Output-Queued Switch Emulation by Fabrics With Limited Memory", in *IEEE Journal on Selected Areas in Communications*, pp. 606–615, May 2003.

[16] S. Iyer, N. McKeown , "Analysis of the Parallel Packet Switch Architecture", in *IEEE/ACM Transactions on Networking*, Vol. 11-2, pp. 314–324, Apr. 2003.

[17] N. McKeown, C. Calamvokis, S. Chuang, "A 2.5Tb/s LCS Switch Core", Hot Chips, Stanford University, Aug. 2001.

[18] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queuing algorithm", *ACM Computer Communication Review (SIGCOMM'89)*, pp. 3–12, 1989.

[19] A. K. Parekh, and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single node case", *IEEE/ACM Transaction on Networking*, Vol. 1, No. 3, pp. 344–357, June 1993.

[20] L. Tassiulas, and A. Ephremides, "Stability Properties of Constrained Queueing Systems and Scheduling Policies for Maximum Throughput in Multihop Radio Networks," IEEE Trans. on Tutomatic Control, 37, pp. 1936–1949, Dec. 1992.

[21] H. J. Kushner, Stochastic Stability and Control, Academic Press. 1967.

[22] G. Fayolle, "On random walks arising in queuing systems: ergodicity and transience via quadratic forms as lyapunov functions - Part I", Queueing Systems, Vol. 5, pp. 167–184, 1989.
[23] N. Chrysos, M. Katevenis, "Weighted Fairness in Buffered Crossbar Scheduling," *Proceedings of the IEEE Workshop on High Performance Switching and Routing*, Torino, Italy, pp. 17–22, June 2003.
[24] K. Yoshigoe and K. J. Christensen, "Design and Evaluation of a Parallel-Polled Virtual Output Queued Switch," *Proceedings of the IEEE 2001 International Conference on Communications*, pp. 112–116, June 2001.

APPENDIX A
ACHIEVING 100% THROUGHPUT FOR AN ARBITRARY SCHEDULING ALGORITHM

Lemma 2: Consider a random variable whose evolution is described by a discrete time markov chain (DTMC) which is aperiodic and irreducible with state vector $Y_n \in N^M$. Suppose that a lower bounded non-negative function, $F(Y_n)$ called a Lyapunov function, $F : N^M \rightarrow R$ exists and suppose that A is a finite subset of N^M, such that $F(Y_n) \leq B' \Rightarrow Y_n \in A, B' \in R$. Then if $E[F(Y_{n+1})|Y_n] < \infty, \forall Y_n$ and there exist $\gamma \in R^+, B \in R^+$, such that,

$$E[F(Y_{n+1}) - F(Y_n)|Y_n] < -\gamma, \forall |Y_n| \geq B \quad (2)$$

then all states of the DTMC are positive recurrent and $\lim_{n\rightarrow\infty} \sup F(Y_n) < \infty$ with probability one.

Proof: This is a straightforward extension of Foster's criteria and follows from [20, 21, 22]. ■

A. Proof for Theorem 1

We will use the above lemma in proving the main theorem of this section.

Theorem 1: Under an arbitrary scheduling algorithm, the buffered crossbar gives 100% throughput with speedup two.

Proof: In the rest of the proof we will assume that all indices i, j, k vary from $1, 2, ..N$. Denote the occupancy of VOQ_{ij} at time n by $Z_{ij}(n)$. Also, let $\tilde{Z}_{ij}$ denote the combined occupancy of the VOQ_{ij} and the crosspoint B_{ij} at time n. By definition, $\tilde{Z}_{ij}(n) = Z_{ij}(n) + B_{ij}(n)$.

Define,

$$f_1(n) = \sum_{i,j} Z_{ij}(n)\Big(\sum_k Z_{ik}(n)\Big) \quad (3)$$

$$f_2(n) = \sum_{i,j} \tilde{Z}_{ij}(n)\Big(\sum_k \tilde{Z}_{kj}(n)\Big) \quad (4)$$

$$F(n) = f_1(n) + f_2(n) \quad (5)$$

Observe that from (3)

$$f_1(n) = \sum_{i,j} Z_{ij}(n)\Big(\sum_k Z_{ik}(n)\Big) = \sum_{i,j,k} Z_{ij}(n)Z_{ik}(n)$$

Denote, $D_{ij}(n)$ = 1 if there is a departure from VOQ_{ij} at time n and zero otherwise. Also, let $A_{ij}(n) = 1$, if there is an arrival to VOQ_{ij} and zero otherwise. Then, $Z_{ij}(n+1) = Z_{ij}(n) + A_{ij}(n) - D_{ij}(n)$. Henceforth, we will drop the time n, from the symbol for $D_{ij}(n)$ and $A_{ij}(n)$, and refer to them as D_{ij} and A_{ij} respectively, since in the rest of the proof, we will only be concerned with the arrivals and departures of cells at time n.

Therefore, $[f_1(n+1) - f_1(n)]$

$$= \sum_{i,j,k} [Z_{ij}(n+1)Z_{ik}(n+1) - Z_{ij}(n)Z_{ik}(n)]$$

Then we get, $[f_1(n+1) - f_1(n)]$

$$= \sum_{i,j,k} (Z_{ij}(n) + A_{ij} - D_{ij})(Z_{ik}(n) + A_{ik} - D_{ik}) - Z_{ij}(n)Z_{ik}(n)$$
$$= \sum_{i,j,k} (A_{ij} - D_{ij})Z_{ik}(n) + (A_{ik} - D_{ik})Z_{ij}(n) + (A_{ij} - D_{ij})(A_{ik} - D_{ik})$$
$$= \sum_{i,j,k} 2(A_{ik} - D_{ik})Z_{ij}(n) + (A_{ij} - D_{ij})(A_{ik} - D_{ik})$$

Since, $|A_{ij} - D_{ij}| \leq 1$, and similarly $|A_{ik} - D_{ik}| \leq 1$, we get $E[f_1(n+1) - f_1(n)]$[9]

$$\leq N^3 + \sum_{i,j,k} 2E[A_{ik} - D_{ik}]Z_{ij}(n) \quad (6)$$

Denote, $\tilde{D}_{ij}(n) = 1$ if there is a departure from the queue which represents the combined occupancy of the VOQ_{ij} and the crosspoint B_{ij}, and zero otherwise. Note that $\tilde{D}_{ij}(n) = 1$ only when there is a departure from the crosspoint B_{ij} to the output at time n, since all departures to the output must occur from the crosspoint. Also recall that the arrival rate to the combined queue, VOQ_{ij} and B_{ij}, is the same as the arrival rate to VOQ_{ij}. So we can write, $\tilde{Z}_{ij}(n+1) = \tilde{Z}_{ij}(n) + A_{ij}(n) - \tilde{D}_{ij}(n)$. Again we will drop the time n, from the symbol for $\tilde{D}_{ij}(n)$ and $A_{ij}(n)$, and refer to them as $\tilde{D}_{ij}$ and A_{ij} respectively.

Then, similar to the derivation in (6), we can derive using (4), $E[f_2(n+1) - f_2(n)]$,

$$\leq N^3 + \sum_{i,j,k} 2E[A_{kj} - \tilde{D}_{kj}]\tilde{Z}_{ij}(n) \quad (7)$$

[9]This is in fact the conditional expectation given knowledge of the state of all queues and crosspoints at time n. For simplicity in the rest of the proof, since we only use the conditional expectation, we will drop the conditional expectation sign and simply use the symbol for expectation, as its meaning is clear.

So from (6) and (7), $E[F(n+1) - F(n)]$

$$\leq 2N^3 + 2\sum_{i,j,k}\Big(E[A_{ik} - D_{ik}]Z_{ij}(n) + E[A_{kj} - \tilde{D}_{kj}]\tilde{Z}_{ij}(n)\Big)$$

$$= 2N^3 + 2\sum_{i,j}\Big(Z_{ij}(n)\sum_k E[A_{ik} - D_{ik}] + \tilde{Z}_{ij}(n)\sum_k E[A_{kj} - \tilde{D}_{kj}]\Big)$$

Re-substituting, $\tilde{Z_{ij}} = Z_{ij} + B_{ij}$, we get $E[f(n+1) - f(n)]$,

$$\leq 2N^3 + 2\sum_{i,j}\Big(Z_{ij}(n)\sum_k E[A_{ik} - D_{ik}] + \big(Z_{ij}(n) + B_{ij}(n)\big)\sum_k E[A_{kj} - \tilde{D}_{kj}]\Big)$$

$$= 2N^3 + 2\sum_{i,j}\Big(Z_{ij}(n)\sum_k E[A_{ik} - D_{ik} + A_{kj} - \tilde{D}_{kj}] + B_{ij}(n)\sum_k E[A_{kj} - \tilde{D}_{kj}]\Big)$$

We can substitute $X_{ij} = \sum_k E[A_{ik} - D_{ik} + A_{kj} - \tilde{D}_{kj}]$ and $Y_j = \sum_k E[A_{kj} - \tilde{D}_{kj}]$ and re-write this as, $E[F(n+1) - F(n)]$

$$\leq 2N^3 + 2\sum_{i,j}\Big(Z_{ij}(n)X_{ij} + B_{ij}(n)Y_j\Big) \quad (8)$$

But, we also have from equation 1,

$$E[C_{ij}(n+1) - C_{ij}(n)] \equiv X_{ij} \quad (9)$$

$$E[\sum_k\Big(\tilde{Z}_{kj}(n+1) - \tilde{Z}_{kj}(n)\Big)] \equiv Y_j \quad (10)$$

In section II it was shown that for a buffered crossbar with speedup two, X_{ij} is strictly negative when $Z_{ij}(n) > 0$ and the traffic is admissible. So the first product term inside the summation sign in equation (8),

$$Z_{ij}(n)X_{ij} \leq 0 \quad (11)$$

Similarly, if the traffic is admissible, then $\sum_k E[A_{kj}] < 1$. Also, when $B_{ij}(n) = 1$, then from (1) and *case 1* of theorem 1 in section II, we know that the output j will receive at least one cell and so at least one cell must have departed one of the crosspoints destined to output j at time n. And so, when the traffic is admissible and $B_{ij}(n) = 1$, then $Y_j < 0$. This implies that the second product term inside the summation sign in equation (8),

$$B_{ij}(n)Y_j \leq 0 \quad (12)$$

In both cases, $Z_{ij}(n)X_{ij}$ and $B_{ij}(n)Y_j$ are equal to zero only if $Z_{ij} = 0$ and $B_{ij} = 0$ respectively. Now we want to use Lemma 2 and show that the whole right hand side of equation (8), is strictly negative. All that needs to be done is to ensure that one of the $VOQs$ Z_{ij} in the summation in equation (8) is large enough so that, $2Z_{ij}(n)X_{ij}$ can negate the positive constant $2N^3$.

In order to show this, let, $\lambda_{max} = \max(\sum_k \lambda_{ik}, \sum_k \lambda_{kj}), i, j \in (1, 2, ..N)$. Choose any $\gamma' > 0$, and let,

$$F \equiv \sum_{ijk} Z_{ij}Z_{ik} + \tilde{Z}_{ij}\tilde{Z}_{kj} >$$

$$N^3\Big[\big(\frac{(1+\gamma')N^3}{1-\lambda_{max}}\big)^2 + \big(1 + \frac{(1+\gamma')N^3}{1-\lambda_{max}}\big)^2\Big] \equiv B$$

where, B corresponds to the constant in Lemma 2. Recall that $\tilde{Z_{ij}} \leq Z_{ij} + 1$. Then the above inequality can only be satisfied if

$$\exists Z_{ij} > \frac{(1+\gamma')N^3}{1-\lambda_{max}}$$

As shown in section II, when $Z_{ij} > 0$,

$$X_{ij} \leq -(2 - 2\lambda_{max}) < -(1 - \lambda_{max})$$

Therefore, we have

$$Z_{ij}X_{ij} < -(1+\gamma')N^3$$

If we substitute this in equation 8, then

$$E[F(n+1) - F(n)] < -2\gamma' N^3, F(n) > B$$

Let γ correspond to the variable in Lemma 2 and set $\gamma = 2\gamma' N^3$. Also it is easy to see that,

$$E[F(n+1)|F(n)] < \infty$$

From Lemma 2, $\lim_{n\to\infty} \sup F(n) < \infty$ with probability one which implies $\lim_{n\to\infty} \sup Z_{ij}(n) < \infty$ with probability one. From definition 2, the scheduling algorithm gives 100% throughput. ■

APPENDIX B
PROOF FOR THEOREM 2

Before we prove the theorem, we will need the following lemmas.

Lemma 3: The slackness $L(c)$ of a cell c waiting on the input side is non-decreasing from time slot to time slot.

Proof: Let $L(c)$ be the slackness of cell c which belongs to VOQ_{ij} at the beginning of a time slot. During the arrival phase, $IT(c)$ can increase by at most one because an arriving cell might be inserted ahead of c in its input priority list. During the departure phase, $OC(c)$ will decrease by at most one. So, $L(c)$ can decrease by at most two in a single time slot.

From Lemma 1, $L(c)$ increases by at least one per scheduling phase. With two scheduling phases per time slot, $L(c)$ increases by at least two. Taking into account arrivals, departures and both scheduling phases, $L(c)$ cannot decrease from time slot to time slot. ■

Lemma 4: The slackness $L(c)$ of a newly arriving cell c is non-negative.

Proof: Consider any cell x that is inserted with a slackness of $L(x)$. Following the arrival phase, $L(x)$ increases by at least one in each of the two scheduling phases. And in the departure phase, $L(x)$ will decrease by one. Therefore, at the end of the time slot, $L(x)$ increases by at least one. For example, if arriving cell x, is inserted with a slackness of zero, then at the end of the time slot, the slackness of cell x will be at least one.

From Lemma 3 and the fact that the slackness of an arriving cell will increase by one at the end of the time slot relative to the slackness of the cell when it arrived, we know that if the slackness of a cell is less than one, then its slackness must have been negative when the cell was inserted. Let t be the first time that an arriving cell is inserted with negative slackness. Consider two cases:

- **Case 1**: If cell c was inserted at the head of the priority list, $IT(c)$ is zero. Since the output cushion is defined as a non-negative value, the slackness of the cell is non-negative when inserted, which contradicts our assumption.
- **Case 2**: If cell c was not inserted at the head of the priority list, cell c must be inserted immediately behind another cell, c', destined to the same output as cell c. Since c' was inserted before time t, it must have been inserted with non-negative slackness. At the end of the time slot cell c' was inserted, its slackness increased by one. From Lemma 3, the slackness of cell c' is still at least one at time t. But since $IT(c) = IT(c') + 1$, and $OC(c) = OC(c')$, then $L(c) = L(c') - 1 \geq 0$. So the slackness of the cell c must also be non-negative when inserted, which again contradicts our assumption.

■

Theorem 2: (Sufficiency) A buffered crossbar with a speedup of two can mimic the restricted PIFO-OQ switch, regardless of the incoming traffic pattern.

Proof: Suppose that the CIOQ switch has successfully mimicked the OQ switch up until time slot $t-1$. Consider the beginning of time slot t. We must show that any cell reaching its departure time is either: (1) already at the output side of the switch, or (2) will be transferred to the output during time slot t.

From Lemma 3 and Lemma 4, we know that a cell always has a non-negative slackness. Therefore, when a cell reaches its departure time (i.e. its output cushion has reached zero), its input thread must also equal zero. This means either: (1) that the cell is already at its output, and may depart on time, (2) that the cell is in the crosspoint buffer or (3) that the cell is simultaneously at the head of its input priority list (because its input thread is zero), and has the earliest departure time (because it has reached its departure time). In case (3), the input scheduling phase is guaranteed to transfer the cell to the crosspoint. Since the cell is in the crosspoint after the input scheduling phase in both cases (2) and (3), and has the earliest departure time, it will be selected in the output scheduling phase. The cell will then reach the output during the time slot, and therefore the cell departs on time. ■

APPENDIX C
PROOF FOR THEOREM 3

In what follows, consider the following virtual finish time assignment policy when a cell arrives to the crosspoint. Assume a cell c which arrives to the crosspoint B_{ij} at time t. Without loss of generality let this be the k^{th} cell from input i to output j.

Case 1: If the $k-1^{th}$ cell is still present in output j of the buffered crossbar, then the output of the buffered crossbar will assign the virtual finish time, $F_i^{k-1} + \frac{1}{\phi_i}$.

Case 2: If the $k-1^{th}$ cell is not present in output j and the k^{th} cell is not transferred to the crosspoint in the scheduling phase immediately after its arrival, then the output of the buffered crossbar will assign a virtual finish time of $F_i^{k-1} + \frac{1}{\phi_i}$.

Case 3: If the $k-1^{th}$ cell is not present in the output j of the buffered crossbar and the k^{th} cell is transferred to the crosspoint immediately after its arrival, then the output of the buffered crossbar will assign the virtual finish time, $V(t) + \frac{1}{\phi_i}$.

We further assume that the buffered crossbar has speedup two and the output picks cells with the smallest virtual finish time from the non-empty crosspoints.

Lemma 5: The virtual finish time of every cell c is the same in the WRR-buffered crossbar switch and the WRR-OQ switch.

Proof: Assume that the buffered crossbar has correctly calculated the virtual finish time of all cells which have arrived to the crosspoints up until time $t-1$ and the outputs have chosen the cells from their crosspoints which have the smallest finish time in every scheduling phase. From the results in section IV, this means that the buffered crossbar with a speedup two, has mimicked the WRR-OQ switch up until time $t-1$. Let t be the first time that the virtual finishing time of a cell calculated is different from the virtual finishing time calculated by the WRR-OQ switch. Consider that cell c which arrives to the crosspoint B_{ij} at time t was incorrectly calculated. Without loss of generality let this be the k^{th} cell from input i to output j. We consider three cases.

Case 1: If the $k-1^{th}$ cell is still present in output j of the buffered crossbar, then this means that it was also present in the WRR-OQ switch when the k^{th} cell arrived. So both the WRR-OQ switch and the output of the buffered crossbar will assign the same virtual finish time, $F_i^{k-1} + \frac{1}{\phi_i}$, which contradicts our assumption.

Case 2: If the $k-1^{th}$ cell is not present in output j and the k^{th} cell is not transferred to the crosspoint

in the scheduling phase immediately after its arrival, then it must have been inserted behind the $k-1^{th}$ cell in the input priority list or it was inserted to the head of the input priority list, but the crosspoint contained the $k-1^{th}$ cell. Since the buffered crossbar switch has mimicked the WRR-OQ switch up until time $t-1$, this means that the $(k-1)^{th}$ cell, was also present in the WRR-OQ switch at time a_i^k. The output of the buffered crossbar assigns a virtual finish time of $F_i^{k-1}+\frac{1}{\phi_i}$ which matches the virtual finish time assigned by the WRR-OQ switch. The assignment is the same, which contradicts our assumption.

Case 3: If the $k-1^{th}$ cell is not present in the output j of the buffered crossbar and the k^{th} cell is transferred to the crosspoint immediately after its arrival, then since the buffered crossbar switch has mimicked the WRR-OQ switch up until time $t-1$, neither switch has cells in the system from input i destined to output j. So both the WRR-OQ switch and the output of the buffered crossbar will assign the same virtual finish time, $V(t)+\frac{1}{\phi_i}$, which again contradicts our assumption.

So the virtual finish time of a cell at time t can also be correctly calculated. ■

The above lemma, and the fact that the WRR-OQ switch is a special case of the restricted PIFO-OQ policy imply the following theorem.

Theorem 3: (Sufficiency) A buffered crossbar can mimic an OQ switch using a weighted round-robin policy with speedup two, regardless of the incoming traffic pattern.

APPENDIX D
PROOF FOR THEOREM 4

Before we prove the theorem, we will need the following lemmas.

Lemma 6: After the modified arrival phase, all cells in the crosspoints B_{ij} will have earlier departure order than any cell queued at input i destined for output j.

Proof: Assume that the above property holds up until time $t-1$. Let t be the first time that any cell c in the crosspoint does not have the earliest departure order as compared to any cell queued at input i destined for output j. At time t, there can be at most one newly arriving cell c to an input. If the arriving cell has a earlier departure order than the cell in the corresponding crosspoint, then the modified arrival phase allows cell c to swap with the cell in the corresponding crosspoint, which contradicts our assumption. ■

Lemma 7: The slackness $L(c)$ of a cell c decreases by at least one in each scheduling phase.

Proof: Since Lemma 6 guarantees that the cell in the crosspoint has the earliest departure order compared with any cell queued in the corresponding input queue, Lemma 1 still holds. ■

Lemma 8: The slackness $L(c)$ of a cell c waiting on the input side is non-decreasing from time slot to time slot.

Proof: Given Lemma 7 the only other difference as compared to Lemma 3 is in the modified arrival phase. Irrespective of whether a swap occurred or not, there is only one newly arriving cell to deal with i.e. if a swap does not occur, then it is a cell which just arrived at the input, else if a swap occurs, then the newly arriving cell is the cell from the swapped crosspoint. The rest of the proof is similar to Lemma 3. ■

Lemma 9: The slackness $L(c)$ of a newly arriving cell c is non-negative.

Proof: As described in Lemma 8, we only need to be concerned about inserting one newly arriving cell to the priority list at the input irrespective of whether a swap occurred or not. The rest of the proof is similar to Lemma 4. ■

Theorem 4: A buffered crossbar can mimic a PIFO-OQ switch (and hence give delay guarantees) with speedup three, regardless of the incoming traffic pattern.

Proof: Given Lemma 8 and Lemma 9, the proof is exactly the same as the proof for Theorem 2. ■

APPENDIX E
PROOF FOR THEOREM 5

Proof: An input can receive at most $p+N-1$ grants over any p consecutive scheduling phases. If the input adds new grants to the tail of a *grant FIFO*, and reads one grant from the head of the grant FIFO in each scheduling phase, then the grant FIFO will never contain more than $N-1$ grants. Each time the input takes a grant from the grant FIFO, it sends the corresponding cell to the set of N crosspoints for its output. Because the grant FIFO is served once per phase, a cell that is granted at scheduling phase p will reach the output crosspoint by phase $p+N-1$.

We need to verify that the per-output buffers in the crossbar never overflow. If the crosspoint scheduler issues a grant at phase p, then the corresponding cell will reach the output crosspoint between phases p and $p+N-1$. Therefore, during scheduling phase p, the only cells which can be in the output crosspoint are cells which were granted between phases $p-N$ to $p-1$. With N buffers per output, the buffers will never overflow, and each cell faces a delay of at most N scheduling phases, or $N/2$ time slots (because $S=2$). ■